Hacking y Pentesting con Python

Módulo 3: Hacking en entornos web con Python.

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Módulo 3: Hacking en entornos web con Python.

PRESENTACIÓN Y OBJETIVOS

Uno de los servicios más extendidos en cualquier infraestructura es precisamente, el servidor web. Se trata de un elemento que brinda acceso a clientes que soportan el protocolo HTTP y permite servir páginas web o transferir información por medio de estructuras de datos. Evidentemente es uno de los puntos de mira de cualquier atacante y por ese motivo no es de extrañar que existan cientos de herramientas orientadas al pentesting y hacking web.

En el caso de Python existen varias librerías interesantes que ayudan a automatizar procesos de detección y en ocasiones, de explotación, en aplicaciones y servidores web.

Objetivos

- Conocer las principales librerías disponibles en Python orientadas al mundo web.
- Aprender a utilizar dichas librerías disponibles en Python para crear rutinas de exploración y extracción de información.
- Aprender a ejecutar rutinas automatizadas para detectar vulnerabilidades en aplicaciones web.

Identificación de vulnerabilidades en aplicaciones web.

Las aplicaciones web constituyen un punto de acceso nada despreciable para un atacante, de hecho, un porcentaje bastante alto de los servidores comprometidos en Internet han sido atacados utilizando vectores relacionados con alguna aplicación web vulnerable o mal configurada. Para automatizar los ataques que se pueden llevar a cabo contra una aplicación web, en Python hay un amplio conjunto de librerías y herramientas que son utilizadas tanto por atacantes como por pentesters profesionales para el descubrimiento de vulnerabilidades.

1. Uso de librerías en Python para entornos web.

Cualquier script que requiera la interacción con servidores web, en la mayoría de los casos requerirá el uso de utilidades que permitan enviar peticiones HTTP y recibir respuestas por parte del servidor web. En Python existen varias librerías que permiten desarrollar clientes HTTP funcionales muy rápidamente y con pocas líneas de código, algunas de estas librerías se encuentran incluidas directamente en la especificación estándar del lenguaje y otras son distribuidas por terceros. A continuación, se detalla el uso básico de algunas de estas librerías y posteriormente se crean scripts que permitan enviar peticiones y recibir respuestas programáticamente.

1.1 Urllib y Urllib2.

Se trata de las librerías más básicas para la creación de clientes HTTP, se encuentran incluidas directamente en el intérprete de Python y no es necesario instalar ningún tipo de dependencia adicional. Son fáciles de utilizar y permiten hacer pruebas rápidas contra servidores web. No obstante, tienen varias limitaciones que dificultan la implementación de algunas de las características descritas en el protocolo HTTP 1.1

```
#python3
Python 3.6.9 (default, Nov  7 2021, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib.request, urllib.parse, urllib.error
>>> response = urllib.request.urlopen('http://www.google.com')
>>> response.getcode()
200
>>> response.geturl()
'http://www.google.com'
>>> for header, value in list(response.headers.items()):
...     print(header+' : '+value)
...
Date : Tue, 24 Mar 2020 16:28:13 GMTExpires : -1
Cache-Control : private, max-age=0
Content-Type : text/html; charset=ISO-8859-1
P3P : CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server : gws
X-XSS-Protection : 0
X-Frame-Options : SAMEORIGIN
Set-Cookie : 1P_JAR=2020-03-24-16; expires=Thu, 23-Apr-2020 16:28:13 GMT;
path=/; domain=.google.com; Secure
Set-Cookie :
NID=200=muR4EcIohpSo37aJ9Ry4w_1SnuePQGCceCp09i217vXizbTbM_ROaZq9YF
NzCZXOqG23AGT9Opf1Zjlz_goDWJpPQsE7sZYhntrywM_JXCzWSQwEfyL6jnwxk3Dd
lECPEN28PygGA8Sk9d7dWDlg4UQPo3XaLgiWJuGHoFjjnSk; expires=Wed, 23-Sep-
2020 16:28:13 GMT; path=/; domain=.google.com; HttpOnly
Accept-Ranges : none
Vary : Accept-Encoding
Connection : close
Transfer-Encoding : chunked
```

Como se puede apreciar, realizar peticiones HTTP contra un recurso en un servidor web es una actividad trivial utilizando urllib, sin embargo, puede ser un poco más complicado cuando es necesario implementar mecanismos de autenticación básica o digest. Por ejemplo, en el caso de implementar autenticación básica sería necesario importar el módulo urllib2 y codificar manualmente la combinación de usuario y clave en Base64.

Actualmente, es más común y recomendable, utilizar las librerías que se listan en los siguientes apartados por cuestiones de simplicidad y rendimiento.

1.2 Librerías Httplib y Httplib2.

Librerías como httplib y httplib2 intentan simplificar el proceso de creación de clientes HTTP y la manipulación de peticiones y respuestas. Además, soportan el protocolo HTTP y HTTPS del lado del cliente de manera automática.

Por otro lado, tanto HTTPLib como HTTPLib2 deben instalarse de forma manual en Python 2.7, sin embargo, se trata de librerías que ya se encuentran incluidas de forma nativa en Python 3.x, pero se han refactorizado los módulos y clases para ordenar mejor el código y evitar las duplicidades que se daban en Python 2.7 al utilizar dichas librerías. El primer ejemplo sobre el uso de esta librería soportado Python3 quedaría reflejado en el script **3-httplib.py**

1.3 Librerías Urllib3 y Request.

La librería urllib3 añade algunas mejoras a las librerías mencionadas anteriormente, especialmente con todo lo relacionado a las características avanzadas del protocolo HTTP 1.1 y recientemente, 2.0.

De forma automática, permite la reutilización de conexiones TCP para generar múltiples peticiones sobre el mismo canal de comunicación, además permite validar los certificados HTTPS emitidos por los servidores y gestiona las redirecciones automáticamente (respuestas HTTP cuyo código es 3xx).

Otra diferencia importante con respecto a las librerías mencionadas antes, es que permite crear un pool de conexiones para gestionarlas todas desde una única referencia de la clase PoolManager

```
python3
Python 3.6.9 (default, Nov  7 2021, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib3
>>> pool = urllib3.PoolManager(10)
>>> res = pool.request('GET', 'http://www.google.com')
>>> res.status
200
>>> res.headers
HTTPHeaderDict({'Date': 'Tue, 24 Mar 2020 16:57:35 GMT', 'Expires': '-1', 'Cache-Control': 'private, max-age=0', 'Content-Type': 'text/html; charset=ISO-8859-1', 'P3P': 'CP="This is not a P3P policy! See g.co/p3phelp for more info."', 'Server': 'gws', 'X-XSS-Protection': '0', 'X-Frame-Options': 'SAMEORIGIN', 'Set-Cookie': '1P_JAR=2020-03-24-16; expires=Thu, 23-Apr-2020 16:57:35 GMT; path=/; domain=.google.com; Secure, NID=200=P23GY8zdtsaucm8Pc3XyWZ6z-BwjVR7vCdC1fgblhFqsWay4xEmftDqcjcPgSVLSYEFkZ2yC9Ph_rlNaV_FNWuAwUsPdL4KJKqF_cqNjKqpurOcxBNnsGkfhRl3lA7kRTPf-rjCpwYoFLPjwHyzPfv4hTqo87miMfhBrt1kcAsU; expires=Wed, 23-Sep-2020 16:57:34 GMT; path=/; domain=.google.com; HttpOnly', 'Accept-Ranges': 'none', 'Vary': 'Accept-Encoding', 'Transfer-Encoding': 'chunked'})
>>> res2 = pool.request('GET', 'http://www.startpage.com')
>>> res3 = pool.request('GET', 'http://www.google.com')
>>> res4 = pool.request('GET', 'http://thehackerway.com')
>>> len(pool.pools)
4
```

El uso de la clase PoolManager es esencial para poder ejecutar y almacenar múltiples peticiones HTTP con urllib3. Otra clase que también merece la pena mencionar, es ProxyManager ya que también permite crear pooles de conexiones, pero utilizando un proxy para manejar las peticiones.

```
>>> proxy = urllib3.ProxyManager('http://127.0.0.1:8008')

>>> proxy.request('GET', 'http://thehackerway.com')

>>> response = proxy.request('GET', 'http://thehackerway.com')
```

Todas las peticiones realizadas con ProxyManager serán enviadas al proxy que se ha definido como primer argumento del constructor de la clase y además, dado que ProxyManager extiende de PoolManager, es posible utilizar los mismos métodos para gestionar un pool de conexiones.

Urllib3 cuenta, además, con dos clases muy interesantes para delimitar a un único dominio las peticiones realizadas desde un pool de conexiones, es decir, que todas las peticiones deben dirigirse a recursos de un dominio determinado. Dicha característica se encuentra soportada por las clases HTTPConnectionPool y HTTPSConnectionPool y permiten controlar las peticiones que puede llevar a cabo un pool de conexiones.

En el caso de que se intente realizar una petición a un recurso de otro dominio, automáticamente la conexión será cortada y se lanzará una excepción indicando el fallo.

```
>>>connection = urllib3.connection_from_url('http://thehackerway.com')
>>> response1 = connection.request('GET',
'https://thehackerway.com/2021/01/04/proximas-entradas/')
>>> response2 = connection.request('GET', 'https://google.com')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python3.6/dist-packages/urllib3/request.py", line 68, in
request
    **urlopen_kw)
  File "/usr/local/lib/python3.6/dist-packages/urllib3/request.py", line 89, in
request_encode_url
    return self.urlopen(method, url, **extra_kw)
  File "/usr/local/lib/python3.6/dist-packages/urllib3/connectionpool.py", line 551,
in urlopen
    raise HostChangedError(self, url, retries)
urllib3.exceptions.HostChangedError:
HTTPSConnectionPool(host='thehackerway.com', port=443): Tried to open a
foreign host with url: https://google.com
```

Como se puede apreciar, cualquier petición contra un dominio distinto del que se ha fijado a la hora de crear una instancia de la clase HTTPConnectionPool, genera una excepción indicando que no se puede abrir una conexión con el host remoto.

Por otro lado, Requests es una librería que se basa en urllib3 y probablemente es la opción más recomendada a la hora de crear proyectos que requieran ejecutar peticiones HTTP contra servidores web. Requests se encarga de encapsular todos los detalles técnicos necesarios para utilizar el protocolo HTTP simplificando la complejidad y reduciendo considerablemente el número de líneas de código necesarias.

Los scripts que utilizan requests no necesitan codificar manualmente los datos que se envían en una petición post de un formulario, conformar los parámetros de una petición por GET, mantener el estado de las peticiones HTTP o parsear las repuestas en formatos web comunes como JSON o XML; estás y muchas otras tareas las realiza automáticamente la librería sin la intervención del programador. Además, soporta la autenticación básica y digest de forma transparente, sin la necesidad de construir manualmente las cabeceras necesarias.

```
#python
Python 3.6.9 (default, Nov  7 2021, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> responseGet = requests.get('http://www.google.com')
>>> responsePost = requests.post('http://www.google.com')
>>> responseGet.status_code
200
>>> responsePost.status_code
405
>>> responsePost.headers
{'Allow': 'GET, HEAD', 'Date': 'Tue, 24 Mar 2020 17:05:31 GMT', 'Content-Type': 'text/html; charset=UTF-8', 'Server': 'gws', 'Content-Length': '1589', 'X-XSS-Protection': '0', 'X-Frame-Options': 'SAMEORIGIN'}
>>> responsePost.request.headers
{'User-Agent': 'python-requests/2.21.0', 'Accept-Encoding': 'gzip, deflate', 'Accept': '*/*', 'Connection': 'keep-alive', 'Content-Length': '0'}
```

El módulo requests cuenta con las funciones necesarias para ejecutar cualquier método HTTP soportado por el protocolo y para controlar los objetos de respuesta. Hay funciones y atributos muy similares a los vistos en las librerías anteriores para acceder a toda la información de peticiones y respuestas que permiten crear clientes HTTP bastante robustos.

2. SPIDERING CONTRA SITIOS WEB.

Además de realizar peticiones HTTP contra sitios web, en la mayoría de los casos es necesario parsear los contenidos de respuesta o incluso, interactuar programáticamente con dichos sitios. Las librerías BeautifulSoup y Mechanize son útiles precisamente para cumplir con esos objetivos, ya que permiten parsear contenidos y navegar por un sitio web de forma automática.

2.1 BeautifulSoup.

En el caso de BeautifulSoup permite extraer cualquier elemento de una página HTML o un documento XML sin necesidad de escribir demasiado código, además soporta parsers tan populares y potentes como lxml y html5lib.

El proceso de instalación es simple, en primer lugar, es necesario descargar la librería desde el sitio oficial ubicado en la siguiente dirección: http://www.crummy.com/software/BeautifulSoup/ posteriormente, basta con ejecutar el script setup.py con el argumento install. También es perfectamente válido utilizar PIP para su instalación tal como se ha visto en librerías anteriores.

El script **3-beautifulsoup.py** demuestra el uso de básico de BeautifulSoup para recuperar el contenido HTML de un sitio web y parsear sus contenidos utilizando lxml como parser.

Como se puede apreciar en dicho script, el primer paso consiste en crear una instancia de la clase **BeautifulSoup** enviando como argumento el contenido HTML de la página y posteriormente, se puede acceder directamente a los elementos de dicha página como atributos del objeto "soup". En este caso lo que se pintará por pantalla contendrá el contenido de las etiquetas title, head y body. Además, para acceder al texto contenido de alguna de las etiquetas del documento, se utiliza el atributo text y de esa forma, solamente se retornará el texto crudo, sin ninguna otra etiqueta HTML anidada.

Otra característica interesante de la librería, es que permite buscar elementos concretos en la estructura del documento, de tal forma que es posible buscar enlaces, formularios o botones en un documento HTML o elementos de un documento XML. El script **3-beautifulsoup2.py** enseña precisamente el caso en el que resulta interesante recuperar todos los enlaces de un sitio web para posteriormente vistarlos, lo que brinda la posibilidad de crear procesos de crawling contra sitios web. La función find_all de una instancia de BeautifulSoup permite encontrar todos los elementos de un tipo determinado y retorna una lista de objetos de la clase Tag, la cual tiene la misma estructura y atributos que la clase BeautifulSoup.

2.2 Mechanize.

Por otro lado, Mechanize es una librería que funciona prácticamente igual que un navegador web, en donde es posible activar botones o enlaces y controlar la navegación, pero evidentemente, desde un script en Python.

Utilizar Mechanize es muy beneficioso desde el punto de vista de un atacante, ya que permite inspeccionar cualquier tipo de elemento en un sitio web como por ejemplo campos de texto u otros elementos de entrada que pueden ser manipulados directamente.

El componente principal de la librería es la clase Browser, la cual cuenta con todos los atributos y funciones necesarias para activar elementos en una página web, controlar el flujo de navegación o incluso, gestionar automáticamente las cookies enviadas por el sitio web. Para instalar la librería, se sigue el mismo procedimiento indicado para instalar cualquier librería en Python. Es posible hacerlo ejecutando la utilidad pip o descargando la última versión desde el sitio web oficial, el cual se encuentra ubicado en la siguiente ruta: http://wwwsearch.sourceforge.net/mechanize/.

Con el script **3-mechanize.py** se abre una conexión con el sitio web “https://www.google.com” utilizando una instancia de la clase Browser y posteriormente se recorren todos los formularios de la página. La función find_control permite encontrar un control especifico en un formulario y la función select_form permite enfocar el navegador en un formulario determinado para poder activarlo posteriormente con la función submit.

Se trata de un script muy simple que detalla las funcionalidades básicas de Mechanize, no obstante, en este caso el sitio web destino detecta que se trata de un bot y rechaza la conexión devolviendo un código de error 403. En tales casos, es posible gestionar de forma automática las cookies enviadas por un sitio web con módulo cookiejar y además, incluir cabeceras personalizadas en el objeto Browser para que las peticiones parezcan ejecutadas desde un navegador web moderno. Estos cambios en el script anterior se pueden ver en el script **3-mechanize2.py**, en donde se podrá comprobar que ahora la petición se realiza sin ningún problema.

Con una instancia de la clase LWPCookieJar se gestionan automáticamente las cookies en un objeto de la clase Browser, además existen algunas funciones adicionales que permiten establecer las propiedades utilizadas por el navegador a la hora de interactuar con aplicaciones web. Por otro lado, en el script **3-mechanize3.py** también se enseña el uso de las funciones para listar los enlaces de un sitio web y activarlos.

2.3 Selenium.

Si bien las librerías mencionadas anteriormente representan una buena alternativa para ejecutar procesos de spidering, no siempre es lo que se busca y puede ser complejo de mantener ya que las estructuras de los sitios web analizados pueden cambiar, haciendo necesario cambiar también los scripts desarrollados.

Una alternativa muy extendida en el punto del pentesting y a la hora de ejecutar pruebas de calidad en el software es utilizar un “WebDriver” que permita realizar la navegación web de forma natural utilizando un navegador como Firefox o Chrome y posteriormente, capturar todas las peticiones enviadas al servidor y para automatizar el proceso de interacción con la aplicación web. Esto es algo que viene muy bien en pentesting web especialmente a la hora de ejecutar pruebas de Fuzzing, de hecho, herramientas como ZAP tienen implementaciones de Selenium integradas con el objetivo de proporcionar una navegación limpia por el sitio web objetivo y capturar todas las peticiones HTTP que se van realizando para posteriormente, ejecutar ataques de forma mucho más ágil.

En el script **3-selenium.py** se puede apreciar el funcionamiento de esta librería con el WebDriver adecuado.

Es importante tener en cuenta que antes de poder utilizar Selenium es necesario descargar el componente que permitirá controlar el navegador web, dicho componente dependerá de si se trata de Firefox o Chrome y evidentemente, el navegador web también se debe encontrar instalado en el sistema donde se ejecuta Selenium.

Se recomienda leer la documentación oficial en donde se explica detalladamente cómo realizar la instalación en el siguiente enlace:

https://selenium-python.readthedocs.io/getting-started.html

3 PUNTOS CLAVE

- Aunque Python cuenta con librerías tales como urllib para crear clientes HTTP, existen otras alternativas que ayudan en la automatización de pruebas.
- Requests es una de las librerías más habituales en proyectos de software en los que se requiere crear clientes HTTP. Es considerada "pythonic" y su uso es sencillo.
- BeautifulSoup es la librería por excelencia a la hora de parsear contenidos web. Resulta muy útil en procesos de scraping.
- Mechanize y Selenium son librerías que ayudan a automatizar la navegación por sitios web, lo que no solamente es útil en procesos de pentesting web, sino que además sirve también para procesos de Q&A.
- Mechanize es una librería mucho más sencilla que Selenium, sin embargo con más limitaciones, especialmente cuando se trata de gestionar contenido dinámico en el lado del cliente como rutinas Javascript.